МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ВСП «ЕКОНОМІКО-ПРАВНИЧИЙ ФАХОВИЙ КОЛЕДЖ
ЗАПОРІЗЬКОГО НАЦІОНАЛЬНОГО УНІВЕРСИТЕТУ»

Циклова комісія математичних дисциплін та інформаційних технологій

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «ПРОЄКТУВАННЯ ТА РОЗРОБКА СИСТЕМИ ПРОДАЖУ ЦИФРОВИХ ТОВАРІВ У TELEGRAM-БОТІ»

Виконав: здобувач освіти 4 курсу, групи К121-22
Спеціальність 121 Інженерія
програмного забезпечення
(шифр і спеціальність)
Данило СКОЧКО
(ім'я та ПРІЗВИЩЕ)
Керівник Дмитро ДРАЄВСЬКИЙ
(ім'я та ПРІЗВИЩЕ)
Рецензент доцент кафедри програмної інженерії ЗНУ, к.т.н., доцент Віталій МУХІН
(посада, вчене звання, науковий ступінь, ім'я та ПРІЗВИЩЕ)

Запоріжжя
2026

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

ВСП «Економіко-правничий фаховий коледж ЗНУ»

Освітньо–кваліфікаційний ступінь фаховий молодший бакалавр

Спеціальність 121 – Інженерія програмного забезпечення
(шифр і назва)

ЗАТВЕРДЖУЮ

Голова циклової комісії математичних дисциплін та інформаційних технологій

_____ Тетяна СМОЛЯНКОВА
(підпис)

“ 05 ” червня 2026 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

СКОЧКО Данилу Олександровичу
(ПРІЗВИЩЕ, ім’я та по– батькові)

1. Тема роботи «Проєктування та розробка системи продажу цифрових товарів у Telegram-боті»

Керівник роботи Драєвський Дмитро Сергійович
(науковий ступінь, вчене звання, прізвище, ім’я, по батькові)

Затверджені наказом ВСП ЕПФК ЗНУ від «01» грудня 2025 р. № 1728-с

2. Строк подання студентом роботи 05.06.2026

3. Вихідні дані до роботи 1. Постановка задачі
2. Перелік літератури

4. Зміст розрахунково– пояснювальної записки (перелік питань, які потрібно розробити)
1. Аналіз предметної області та вибір стеку технологій
2. Проєктування бази даних та архітектури системи
3. Програмна реалізація вебдодатка

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)
презентація до захисту

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання 01.12.2025

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи:		
	Постановка задачі	Грудень 2025	виконано
2.	Збір вихідних даних, обробка методичних		
	та теоретичних джерел	Січень 2026	виконано
3.	Розробка першого розділу:		
	Огляд інформаційних систем та технологій для їхнього створення	Лютий 2026	виконано
4.	Розробка другого розділу:		
	Розробка проєкту інформаційної системи та вибір технологій	Березень 2026	виконано
5.	Розробка третього розділу:		
	Розробка застосунку	Квітень 2026	виконано
6.	Оформлення і нормоконтроль кваліфікаційної роботи та перевірка на плагіат	Травень 2026	виконано
7.	Захист кваліфікаційної роботи	16.06.2026	виконано

Здобувач освіти _____ Данило СКОЧКО
(підпис) (ім’я ПРІЗВИЩЕ)

Керівник роботи _____ Дмитро ДРАЄВСЬКИЙ
(підпис) (ім’я ПРІЗВИЩЕ)

Нормоконтроль пройдено

Нормоконтролер _____ Юлія БОРИСОВСЬКА
(підпис) (ім’я ПРІЗВИЩЕ)

РЕФЕРАТ

Кваліфікаційна робота: 40 сторінок, 7 рисунків, 20 джерел.

Кваліфікаційна робота присвячена проєктуванню та програмній реалізації високонавантаженої інформаційної системи для електронної комерції у вигляді Telegram-бота. Об’єктом дослідження виступає процес автоматизації продажу цифрового контенту через інтерактивні інтерфейси месенджерів, тоді як предметом є відповідні програмні засоби, архітектурні патерни та алгоритми для розробки асинхронних додатків на базі Telegram API. Основна мета роботи полягала у створенні надійної системи торгівлі з інтегрованою панеллю адміністрування та вбудованими механізмами захисту від автоматизованого спаму.

У процесі розробки застосовувалися методи об'єктно-орієнтованого аналізу для побудови структури класів на основі патерну Data Access Object. Для оптимізації структури зберігання даних використовувалися методи реляційної алгебри, а маршрутизація користувацьких діалогів реалізована за допомогою теорії скінченних автоматів (FSM).

У результаті виконання роботи створено повнофункціональну програмну систему на мові Python з використанням асинхронного фреймворку aiogram. Для забезпечення максимальної швидкодії спроєктовано локальну базу даних SQLite із застосуванням методів часткової денормалізації, що дозволило прискорити доступ до користувацьких кошиків без використання ресурсомістких ORM-систем. Додатково впроваджено систему графічної верифікації клієнтів (генерація капчі), модуль візуалізації статистики продажів на базі matplotlib та механізм динамічної конфігурації без перезапуску сервера. Розроблена система є повністю готовою до розгортання та практичного використання у сфері роздрібної торгівлі.

TELEGRAM-БОТ, ЕЛЕКТРОННА КОМЕРЦІЯ, PYTHON, AIOGRAM, SQLITE, АСИНХРОННЕ ПРОГРАМУВАННЯ, СКІНЧЕННІ АВТОМАТИ, ПАТЕРНИ ПРОЄКТУВАННЯ

SUMMARY

The qualification work contains: 40 pages, 7 figures, 20 references.

The qualification work is devoted to the design and software implementation of a high-load information system for e-commerce in the form of a Telegram bot. The object of the study is the automation process of selling digital content through interactive messenger interfaces, while the subject includes the software tools, architectural patterns, and algorithms required for developing asynchronous applications based on the Telegram API. The primary objective was to create a reliable trading system with an integrated administration panel and built-in protection mechanisms against automated spam.

During the development process, object-oriented analysis methods were applied to build a class structure based on the Data Access Object pattern. Relational algebra methods were used to optimize the data storage structure, and user dialogue routing was implemented utilizing finite state machine (FSM) theory.

As a result, a fully functional software system was developed using the Python programming language and the asynchronous aiogram framework. To ensure maximum performance, a local SQLite database was designed using partial denormalization methods, accelerating access to user shopping carts without relying on resource-intensive ORM systems. Additionally, the project features a graphical client verification system (captcha generation), a sales statistics visualization module based on matplotlib, and a dynamic configuration mechanism that operates without server restarts. The developed system is completely ready for deployment and practical use in the retail sector.

TELEGRAM BOT, E-COMMERCE, PYTHON, AIOGRAM, SQLITE, ASYNCHRONOUS PROGRAMMING, FINITE STATE MACHINES, DESIGN PATTERNS

ЗМІСТ

Завдання на кваліфікаційну роботу студенту 2
Реферат 4
Summary 6
Вступ 8
1 Дослідження технологій та аналіз інформаційних процесів продажу в середовищі месенджерів 10
1.1. Характеристика сфери цифрових продажів та використання прикладних інтерфейсів 10
1.2 Порівняльна оцінка існуючих архітектурних підходів до створення торгових ботів 11
1.3 Специфікація вимог до функціоналу та програмної стійкості додатка 13
1.4 Обґрунтування інструментальних засобів та середовища розробки 14
2 Системне проєктування логіки та структури даних чат-бота 18
2.1 Формування концептуальної архітектури програмного комплексу 18
2.2 Розробка логічної моделі бази даних та підходів до її денормалізації..... 19
2.3 Алгоритмізація маршрутизації діалогів за допомогою скінченних автоматів 21
2.4 Об'єктно-орієнтоване проєктування моделей взаємодії з даними 23
2.5 Проєктування підсистем резервного копіювання та верифікації клієнтів 25
3 Практична реалізація асинхронних модулів та інтерфейсів системи 27
3.1 Програмування ядра бота та обробки вхідних подій 27
3.2 Розробка клієнтського функціоналу для навігації та управління кошиком ... 28
3.3 Реалізація механізмів збору контактної інформації при оформленні замовлень 29
3.4 Програмне забезпечення функцій панелі адміністрування 31
3.5 Створення підсистеми генерації візуальної аналітики 32
4 Експлуатаційне тестування та впровадження програмного продукту 34
4.1 Верифікація роботи ключових алгоритмів та інтерфейсів 34
4.2 Оцінка захищеності системи від некоректного введення даних 35
4.3 Підготовка середовища та розгортання бота на віддаленому сервері..... 36
Висновки 37
Перелік використаних джерел 39

ВСТУП

Сучасний етап розвитку електронної комерції характеризується стійкою тенденцією до мінімізації кроків, які користувач має зробити від моменту пошуку товару до його оплати [1]. Традиційні веб-рішення часто вимагають переходу за зовнішніми посиланнями, обов'язкової реєстрації через електронну пошту та тривалої навігації по інтерфейсу браузера. Це створює додаткові бар'єри та призводить до зниження загальної конверсії. Альтернативним підходом, що активно розвивається і демонструє високу ефективність, є використання мікросервісних архітектур на базі популярних месенджерів. Застосування Telegram API дозволяє делегувати процеси автентифікації клієнта інфраструктурі месенджера, забезпечуючи безшовний доступ до каталогу товарів [2]. Розробка спеціалізованих чат-ботів для продажу дозволяє малому та середньому бізнесу значно скоротити витрати на розробку і підтримку класичних інтернет-магазинів, водночас забезпечуючи високу швидкість обробки замовлень. З огляду на це, розробка оптимізованих, стійких до навантажень систем автоматизації продажів у середовищі месенджерів є надзвичайно актуальною інженерною та бізнес-задачею.

Основною метою даної кваліфікаційної роботи є проєктування та програмна реалізація високонавантаженої інформаційної системи для електронної комерції у форматі Telegram-бота з інтегрованою панеллю адміністрування та механізмами захисту від фроду. Відповідно до поставленої мети було сформовано комплекс завдань. Насамперед вимагалося провести ґрунтовний аналіз існуючих архітектурних підходів до розробки чат-ботів та аргументовано обрати технологічний стек. Наступним етапом стало проєктування структури реляційної бази даних, яка б враховувала специфіку зберігання станів користувачів та забезпечувала мінімальний час відгуку системи. Ключовим практичним завданням стала реалізація асинхронного бекенду для обробки великих масивів вхідних запитів без блокування

основного потоку виконання. Разом з тим необхідно було розробити модуль генерації статистичних звітів та повноцінну панель управління каталогом для адміністратора, а також впровадити механізми безпеки, зокрема систему верифікації клієнтів та алгоритми автоматичного резервного копіювання даних.

Об’єктом дослідження в рамках даної роботи виступає безпосередньо процес автоматизації продажу цифрового контенту та фізичних товарів через інтерактивні інтерфейси месенджерів. Водночас предметом дослідження є програмні засоби, архітектурні патерни та алгоритми, необхідні для розробки асинхронних клієнт-серверних додатків на базі Telegram API.

Практичне значення отриманих результатів полягає у тому, що створена програмна система є повністю функціональним і самостійним IT-продуктом, готовим до впровадження у реальні процеси роздрібної торгівлі. Запропоновані архітектурні рішення, зокрема відмова від важких систем об'єктно-реляційного відображення на користь оптимізованих SQL-запитів та методів часткової денормалізації бази даних, дозволяють розгортати систему на бюджетних серверах без втрати загальної продуктивності. Вбудовані інструменти аналітики та динамічного налаштування конфігурації значно спрощують адміністрування програмного комплексу без необхідності залучення профільних технічних спеціалістів для його щоденної підтримки.

1 ДОСЛІДЖЕННЯ ТЕХНОЛОГІЙ ТА АНАЛІЗ ІНФОРМАЦІЙНИХ ПРОЦЕСІВ ПРОДАЖУ В СЕРЕДОВИЩІ МЕСЕНДЖЕРІВ

1.1. Характеристика сфери цифрових продажів та використання прикладних інтерфейсів

Еволюція електронної комерції протягом останнього десятиліття демонструє кардинальну зміну парадигми взаємодії між постачальником послуг та кінцевим споживачем. Якщо на ранніх етапах розвитку індустрії основним каналом збуту були монолітні веб-сайти, а згодом - спеціалізовані мобільні додатки, то сьогодні спостерігається стійкий перехід аудиторії до концепції розмовної комерції (Conversational Commerce) [3]. Цей феномен зумовлений явищем так званої втоми від додатків (App Fatigue), коли користувачі відмовляються завантажувати та встановлювати нове програмне забезпечення для здійснення разових або нерегулярних покупок. Класичний процес придбання товару в інтернет-магазині часто супроводжується високим рівнем когнітивного навантаження та технічного тертя. Користувачеві необхідно відкрити браузер, дочекатися завантаження важких скриптів та графічних елементів інтерфейсу, пройти багатокрокову процедуру реєстрації, підтвердити електронну пошту та розібратися в унікальній навігації конкретного сайту. Кожен із цих етапів діє як фільтр, що неминуче призводить до відтоку потенційних клієнтів і зниження загального показника конверсії. Натомість інтеграція бізнес-процесів безпосередньо в екосистеми популярних месенджерів дозволяє повністю нівелювати більшість із цих бар'єрів.

Використання прикладних програмних інтерфейсів, зокрема Telegram API, надає розробникам можливість створювати повноцінні торгові майданчики всередині середовища, яке вже встановлене на пристрої клієнта. Оскільки користувач попередньо авторизований у системі месенджера за номером телефону, відпадає необхідність у створенні додаткових механізмів

реєстрації та управління паролями. Взаємодія з програмним ботом відбувається у звичному для клієнта текстовому чат-інтерфейсі за допомогою спеціальних кнопок швидкої відповіді (Inline та Reply клавіатури). Це створює ілюзію прямого діалогу, що психологічно знижує бар'єр перед здійсненням покупки. Крім того, розгортання торгової площадки у форматі чат-бота дозволяє бізнесу радикально скоротити витрати на розробку та подальшу підтримку кросплатформності. Клієнтський додаток Telegram вже оптимізований під усі існуючі мобільні операційні системи (iOS, Android) та десктопні середовища (Windows, macOS, Linux). Відповідно, єдина кодова база бекенд-сервера здатна обслуговувати користувачів незалежно від типу їхнього пристрою, розміру екрана чи версії операційної системи, що робить розробку ботів економічно вигіднішою альтернативою порівняно зі створенням нативних мобільних додатків.

1.2 Порівняльна оцінка існуючих архітектурних підходів до створення торгових ботів

На сучасному ринку програмної інженерії сформувалося кілька кардинально різних підходів до створення торгових систем у середовищі месенджерів, кожен з яких має свої архітектурні переваги та технічні обмеження. Перший і найбільш поширений серед малого бізнесу підхід базується на використанні готових SaaS-рішень та конструкторів без коду (No-code платформ). Такі хмарні сервіси дозволяють зібрати базового чат-бота з візуальних блоків шляхом налаштування тригерів та відповідей через графічний інтерфейс. Незважаючи на вкрай низький поріг входження та можливість запустити проєкт за кілька годин, цей підхід є неприйнятним для побудови складних та безпечних систем. Конструктори накладають жорсткі обмеження на бізнес-логіку, не дозволяють імплементувати нестандартні математичні розрахунки в кошику, унеможливлюють інтеграцію кастомних

платіжних шлюзів та створюють жорстку прив'язку до постачальника послуги (Vendor Lock-in). Найкритичнішим недоліком є те, що вся база даних клієнтів, їхні персональні дані та історія замовлень фізично зберігаються на серверах сторонньої компанії, що порушує принципи інформаційної безпеки та ускладнює дотримання політик конфіденційності.

Другий підхід пов'язаний із використанням новітньої технології Web App (Mini Apps), яка дозволяє відкривати повноцінні веб-сторінки безпосередньо у вбудованому браузері месенджера. Цей метод надає розробникам повну свободу у створенні багатого візуального інтерфейсу з використанням сучасних фронтенд-фреймворків. Проте архітектура такого рішення вимагає одночасної розробки як клієнтської частини (HTML, CSS, JavaScript), так і серверного API для обробки запитів від цієї сторінки. Це суттєво збільшує обсяг кодової бази, підвищує вимоги до кваліфікації розробників та загальну вартість проєкту. Крім того, завантаження веб-додатку потребує стабільного та швидкого інтернет-з'єднання, що може погіршити користувацький досвід у зонах зі слабким покриттям мережі. Третій підхід, який було обрано як найбільш оптимальний для даного дослідження, полягає у нативній розробці серверного чат-бота на мові програмування загального призначення. У цій архітектурі взаємодія з клієнтом відбувається виключно через обмін форматованими текстовими повідомленнями та натискання кнопок, які генерують зворотні виклики (Callback Queries). Такий метод гарантує максимально швидкий відгук системи навіть при низькій швидкості інтернету, забезпечує мінімальне споживання мережевого трафіку та залишає розробнику повний контроль над алгоритмами обробки, зберігання та захисту інформації на власному ізольованому сервері.

1.3 Специфікація вимог до функціоналу та програмної стійкості додатка

Проєктування будь-якої інформаційної системи вимагає попереднього формування чіткої специфікації функціональних та нефункціональних вимог, які базуються на міжнародних стандартах якості програмного забезпечення. З точки зору функціональності, ядро програмного комплексу повинно реалізувати повний цикл електронної комерції. Це включає можливість динамічного перегляду каталогу з ієрархічним поділом на категорії, додавання обраних товарних позицій до віртуального кошика зі збереженням їх кількості, а також покрокове оформлення замовлення із обов'язковим збором контактних даних та інформації щодо логістики. Важливим функціональним компонентом є розробка закритої панелі адміністрування, яка повинна бути інтегрована безпосередньо в інтерфейс бота. Ця підсистема має надавати уповноваженим користувачам інструменти для оперативного управління товарним асортиментом (додавання, редагування, видалення товарів), перегляду детальної історії транзакцій, зміни статусів виконання замовлень та формування аналітичних звітів у візуальному форматі. Додатковою вимогою є імплементація превентивних механізмів захисту, зокрема системи генерації випадкової графічної капчі, яка перешкоджатиме автоматизованим скриптам перевантажувати базу даних фіктивними користувачами.

У розрізі нефункціональних вимог критичним пріоритетом виступає здатність серверної частини обробляти пікові навантаження без деградації продуктивності та втрати даних. Механізм взаємодії серверів Telegram з додатком розробника має специфічні особливості. Під час проведення рекламних кампаній або масових розсилок, сервер може одночасно отримувати тисячі оновлень (Updates), що створює так званий шторм повідомлень. Якщо архітектура додатка передбачає синхронну обробку, кожен такий запит буде блокувати виконання програми на час звернення до бази даних або очікування відповіді від файлової системи. Це неминуче призведе

до накопичення черги запитів, спрацьовування мережевих тайм-аутів і, як наслідок, блокування бота серверами Telegram через помилку перевищення ліміту підключень. Відповідно, ключовою нефункціональною вимогою до системи є повна асинхронність усіх операцій вводу-виводу. Крім того, система повинна відповідати критеріям високої ремонтопридатності та масштабованості, що вимагає винесення всіх змінних конфігурацій (токени доступу, ідентифікатори адміністраторів, налаштування відображення) у зовнішні файли параметрів для можливості їх гарячої заміни без зупинки основного процесу.

1.4 Обґрунтування інструментальних засобів та середовища розробки

1.4.1. Вибір базової мови та асинхронного фреймворку

Вирішення комплексної проблеми неблокуючого вводу-виводу вимагало ретельного аналізу сучасних мов програмування та їхніх екосистем. Для розробки серверної частини було обрано інтерпретовану мову загального призначення Python [4]. Цей вибір обґрунтований наявністю широкого спектра бібліотек для роботи з мережевими протоколами, обробки даних та високою швидкістю створення прототипів. Стандарт оформлення коду PEP 8 [5] регламентує єдиний стиль написання Python-коду, що суттєво підвищує читабельність та підтримуваність проєкту. Проте ключовою архітектурною проблемою мови Python є наявність механізму глобального блокування інтерпретатора (Global Interpreter Lock, GIL), який фізично забороняє паралельне виконання байт-коду кількома потоками операційної системи в межах одного процесу. Через це використання класичних багатопотокових фреймворків, таких як pyTelegramBotAPI (telebot), є неефективним для

високонавантажених ботів. При надходженні великої кількості запитів створення нового системного потоку для кожного з них призводить до швидкого вичерпання лімітів оперативної пам'яті сервера, а GIL зводить нанівець усі переваги багатоядерних процесорів, змушуючи потоки конкурувати за право виконання.

У якості рішення було обрано перехід до подієво-орієнтованої парадигми з використанням асинхронного фреймворку aiogram [6]. На відміну від багатопотокових аналогів, цей фреймворк базується на стандартній бібліотеці asyncio [7], яка реалізує концепцію єдиного циклу подій (Event Loop). У такій архітектурі всі запити користувачів виконуються в єдиному системному потоці як корутини. Коли програма стикається з операцією, що вимагає часу на очікування (наприклад, виконання SQL-запиту, читання файлу зображення з диска або відправка HTTP-запиту), поточна корутина віддає контроль управління назад у цикл подій. Цикл миттєво перемикається на обробку наступного користувацького запиту, замість того щоб простоювати. Це дозволяє одному ядру процесора ефективно підтримувати тисячі одночасних з'єднань. Інструментарій aiogram також надає потужну підсистему диспетчеризації та фільтрації подій, що спрощує процес маршрутизації вхідних повідомлень та реалізацію складних багатокрокових діалогів на базі скінченних автоматів.

1.4.2. Оцінка рішень для організації локального зберігання даних

Проєктування надійної підсистеми зберігання даних вимагає балансу між продуктивністю, споживанням ресурсів та складністю адміністрування. Розгортання повноцінних систем управління реляційними базами даних клієнт-серверного типу, таких як PostgreSQL або MySQL, забезпечує високу стійкість до конкурентних транзакцій, проте створює значне навантаження на апаратні ресурси. Крім того, обмін даними між додатком на Python та окремим

сервером БД вимагає постійного встановлення TCP-з'єднань, серіалізації даних та передачі їх через мережевий стек операційної системи, що додає невідворотні затримки на рівні мілісекунд для кожного запиту. У контексті Telegram-бота, який має реагувати на дії користувача миттєво, сума таких затримок може негативно вплинути на загальну швидкість роботи інтерфейсу. З огляду на ці фактори, було прийнято інженерне рішення використати легку реляційну СУБД SQLite [8], яка функціонує як вбудована бібліотека та зберігає всю структуру бази даних, таблиці, індекси та самі дані у єдиному локальному файлі на жорсткому диску.

Окремим аспектом оптимізації стало свідоме відхилення від використання популярних систем об'єктно-реляційного відображення (ORM), таких як SQLAlchemy або Peewee. Незважаючи на те, що ORM-інструменти спрощують написання коду шляхом мапінгу об'єктів мови програмування на таблиці бази даних, вони генерують значний обчислювальний оверхед. Процес трансляції Python-об'єктів в абстрактне синтаксичне дерево, генерація SQL-коду, перевірка діалектів та зворотна трансформація отриманих рядків в об'єкти вимагає значних витрат процесорного часу. В архітектурі розроблюваної системи реалізовано пряму взаємодію з підсистемою зберігання через стандартний модуль sqlite3 [9]. Виконання "сирих" параметризованих SQL-запитів [10] не лише унеможливлює ризики SQL-ін'єкцій, але й гарантує максимально можливу пропускну здатність бази даних. Такий мінімалістичний підхід робить розроблений програмний комплекс вкрай невимогливим до серверного обладнання, дозволяючи розгортати його на найдешевших віртуальних приватних серверах без ризику падіння продуктивності.

1.4.3. Інтеграція додаткових бібліотек для побудови аналітики та захисту

Повноцінне функціонування комерційного бота неможливе без забезпечення надійних механізмів збору, валідації та візуалізації даних. Для

відокремлення бізнес-логіки від параметрів середовища виконання було імплементовано патерн управління конфігураціями за допомогою стандартної бібліотеки configparser [11]. Усі критичні змінні, від токенів доступу до налаштувань відображення графіків та логістичних опцій, зберігаються у зовнішньому INI-файлі. Це архітектурне рішення перетворює систему на динамічно керований комплекс, де адміністратор може керувати параметрами роботи безпосередньо через редагування текстового файлу, що не вимагає перекомпіляції коду або перезапуску основного системного процесу. З метою захисту підсистеми зберігання від забруднення некоректними даними під час формування замовлень, в архітектуру інтегровано спеціалізований модуль phonenumbers [12]. Ця бібліотека забезпечує суворий парсинг та нормалізацію текстового вводу, порівнюючи введені телефонні номери з актуальними світовими планами нумерації, що виключає можливість збереження недійсних або форматно неправильних контактів.

Забезпечення захисту від автоматизованих скриптів, які можуть здійснювати атаки типу відмову в обслуговуванні (DDoS) на рівні логіки додатку, реалізовано шляхом інтеграції бібліотеки captcha [13]. Даний модуль відповідає за генерацію випадкових растрових зображень з контрольними символами, які піддаються геометричним спотворенням для ускладнення їх розпізнавання системами оптичного розпізнавання символів (OCR). Цей бар'єр діє як надійний фільтр на етапі ініціалізації сесії користувача. Нарешті, для задоволення вимог адміністраторів щодо бізнес-аналітики, було використано потужну математичну бібліотеку matplotlib [14]. Її можливості дозволяють системі агрегувати фінансові та кількісні показники безпосередньо з бази даних і генерувати високоякісні графіки та гістограми в оперативній пам'яті сервера (через віртуальні буфери обміну). Згенеровані зображення миттєво відправляються в чат адміністратору, що позбавляє необхідності розгортати складні зовнішні аналітичні панелі, такі як Grafana або Kibana, суттєво спрощуючи архітектуру проєкту.

2 СИСТЕМНЕ ПРОЄКТУВАННЯ ЛОГІКИ ТА СТРУКТУРИ ДАНИХ ЧАТ-БОТА

2.1 Формування концептуальної архітектури програмного комплексу

При проєктуванні архітектури сучасних інформаційних систем, орієнтованих на взаємодію через месенджери, критично важливим є розуміння принципів роботи проміжних прикладних інтерфейсів. На відміну від класичних веб-додатків, де браузер клієнта напряму звертається до сервера розробника, архітектура Telegram-бота передбачає обов'язкову наявність потужного посередника у вигляді хмарної інфраструктури самого месенджера. Усі повідомлення, натискання інлайн-кнопок та медіафайли спочатку потрапляють на сервери Telegram, проходять первинну обробку, і лише після цього ретранслюються на сервер додатку. Концептуальна схема взаємодії компонентів системи та потоків даних наведена на рисунку 2.1.

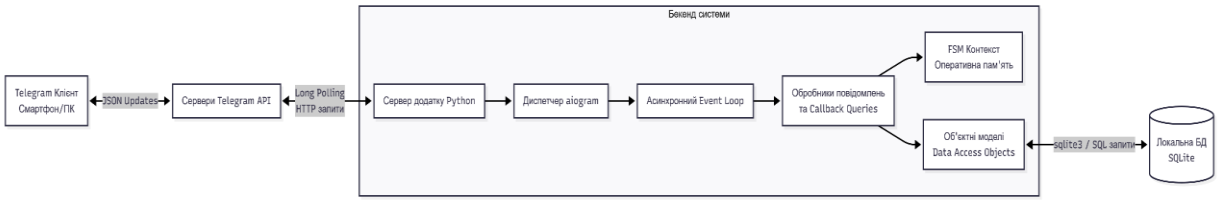


Рисунок 2.1 – Концептуальна схема взаємодії компонентів системи та потоків даних

Для реалізації механізму отримання подій у даній роботі було спроєктовано систему на базі методу тривалого опитування [15]. Цей підхід передбачає, що серверний додаток відкриває мережеве з'єднання з API Telegram та підтримує його активним до моменту надходження масиву нових подій у форматі JSON. Вибір саме цього методу обґрунтований тим, що він

гарантує стабільну роботу додатку навіть в умовах нестабільного мережевого з'єднання або використання NAT-маршрутизації на стороні хостинг-провайдера. Отримані структуровані дані негайно передаються до асинхронного диспетчера, який виконує первинну маршрутизацію. Завдяки використанню єдиного циклу подій, диспетчер не створює нових системних потоків для кожного користувача, а ставить завдання у неблокуючу чергу виконання, що дозволяє одному процесу обробляти запити від тисяч клієнтів одночасно.

2.2 Розробка логічної моделі бази даних та підходів до її денормалізації

Центральним компонентом побудованої системи електронної комерції є підсистема зберігання даних, яка відповідає за персистентність інформації про клієнтів, товарну номенклатуру та історію транзакцій. Для даного проєкту розроблено реляційну схему, що базується на чотирьох основних взаємопов'язаних сутностях, логічна структура яких проілюстрована на рисунку 2.2 у вигляді ER-діаграми (Entity-Relationship diagram).

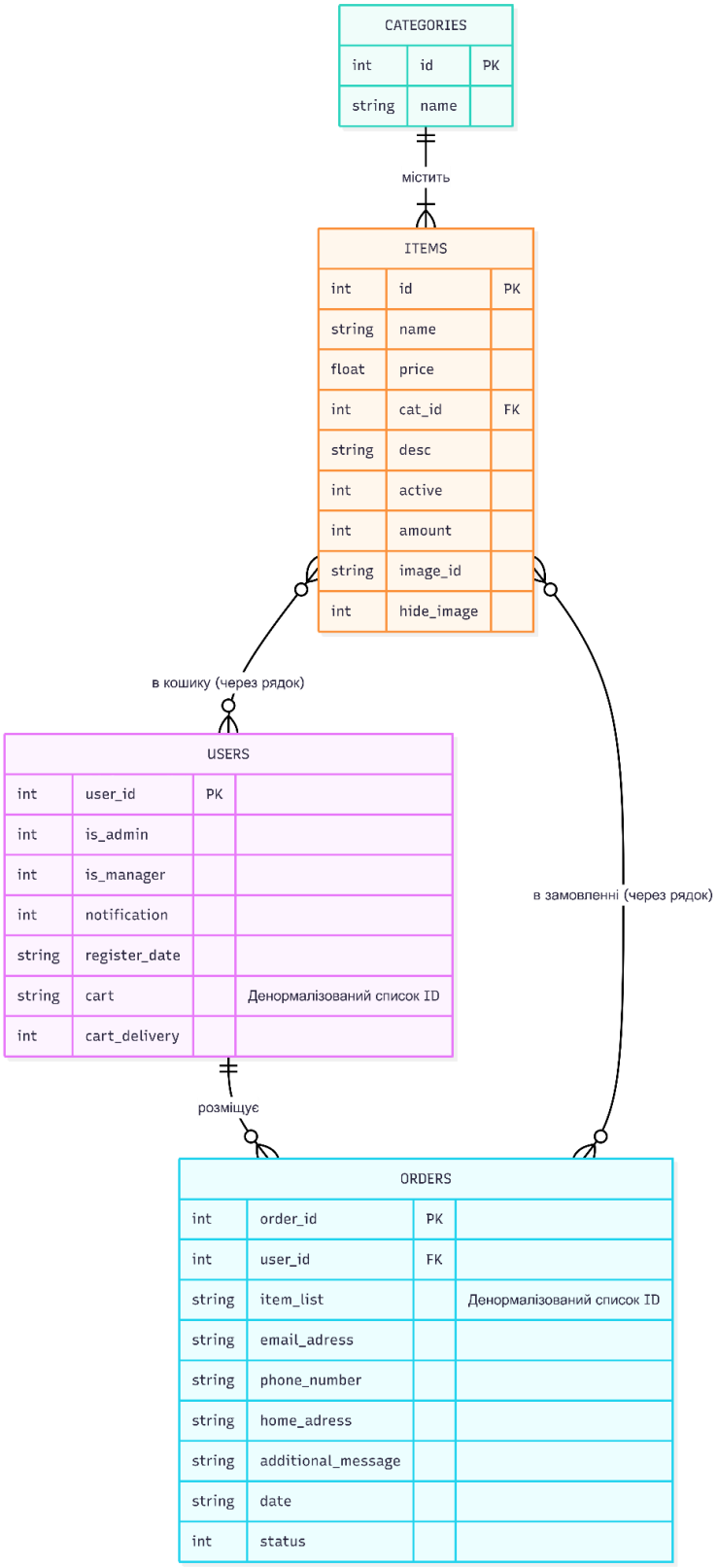


Рисунок 2.2 – ER-діаграма логічної структури бази даних

У класичній теорії проєктування баз даних вимагається суворе дотримання правил нормалізації [16]. Зокрема, перша нормальна форма забороняє зберігання списків або неатомарних значень у межах одного поля

таблиці. Відповідно до цих канонів, для реалізації механізму віртуального кошика користувача необхідно було б створити окрему транзитну таблицю, яка б містила зовнішні ключі ідентифікатора клієнта та ідентифікатора обраного товару. Проте, в умовах роботи високонавантаженого чат-бота, де кожна зміна кількості товарів вимагає миттєвого перемальовування інтерфейсу, виконання складних багаторівневих операцій об'єднання таблиць стало б вузьким місцем продуктивності, створюючи надмірне навантаження на підсистему вводу-виводу сервера.

З огляду на необхідність максимального прискорення часу відгуку системи, було прийнято свідоме інженерне рішення застосувати метод контрольованої денормалізації бази даних. У таблиці користувачів створено єдине текстове поле, яке відповідає за зберігання поточного стану кошика. Ідентифікатори обраних товарів серіалізуються у це поле у вигляді текстового рядка з розділювачами. При додаванні нового товару система програмно витягує цей рядок, конкатенує новий ідентифікатор і виконує елементарний запит на оновлення одного запису. Аналогічний архітектурний патерн застосовано і для таблиці замовлень, де повний перелік придбаних позицій також фіксується у вигляді текстового масиву. Така оптимізація радикально зменшила кількість звернень до підсистеми зберігання. Процес десеріалізації масивів, підрахунку кількості однакових позицій та обчислення загальної вартості замовлення повністю перенесено на рівень прикладної логіки додатку мовою Python. Враховуючи високу швидкість виконання операцій з рядками у цій мові програмування, такий розподіл навантаження між базою даних та логічним ядром системи забезпечує оптимальний баланс між споживанням оперативної пам'яті та процесорного часу.

2.3 Алгоритмізація маршрутизації діалогів за допомогою скінченних автоматів

Процес взаємодії користувача з торговою системою є динамічним і вимагає алгоритмізації складних послідовностей дій. Оформлення замовлення

вимагає від клієнта поетапного введення різнотипної інформації: згоди на обробку даних, введення адреси доставки, зазначення контактного номера телефону та формування додаткових коментарів для кур'єра. Оскільки протокол HTTP та базовий API Telegram функціонують без збереження стану сесії, сервер об'єктивно не володіє інформацією про те, на якому саме етапі діалогу знаходиться конкретний клієнт, розглядаючи кожне повідомлення як незалежну подію. Для вирішення цієї проблеми у програмному коді реалізовано математичну модель скінченного автомата [17]. Ця концепція передбачає, що користувацька сесія може перебувати виключно в одному з наперед визначених станів, а транзакція до наступного стану можлива лише за умови отримання валідної вхідної події. Граф переходів скінченного автомата при оформленні замовлення схематично представлено на рисунку 2.3.

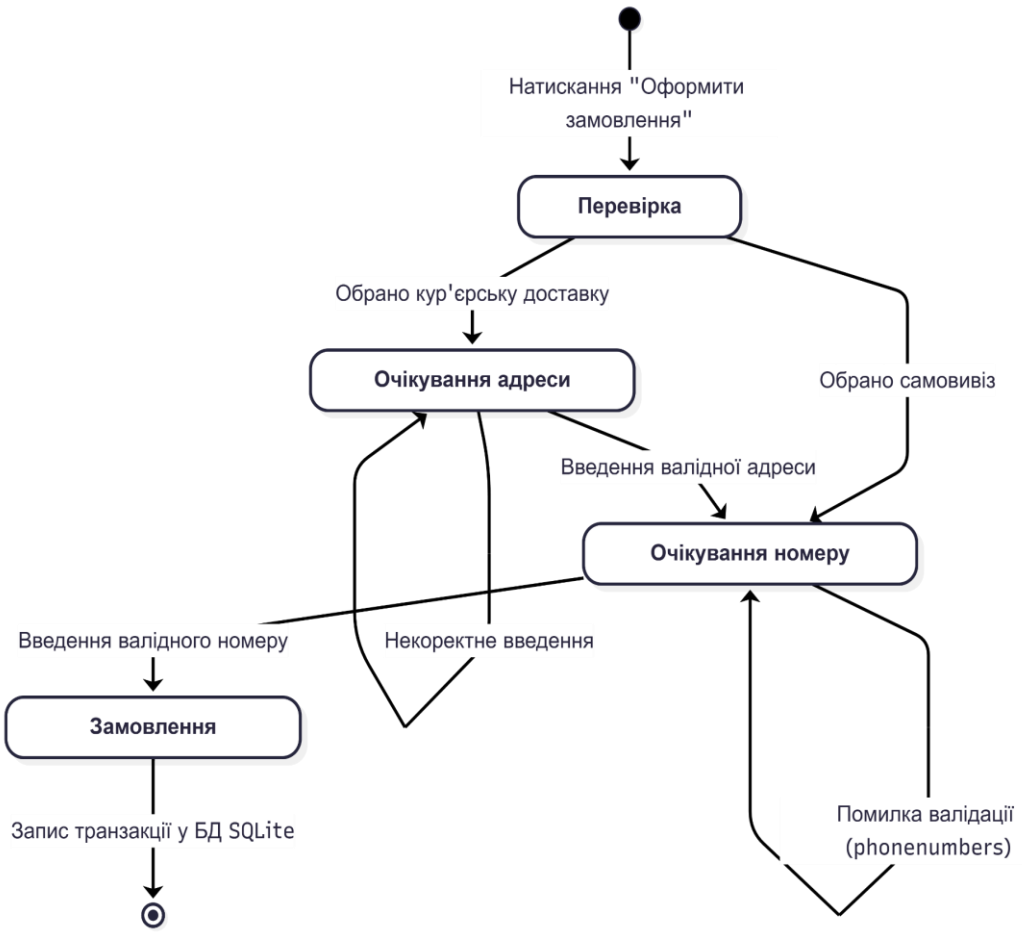


Рисунок 2.3 - Граф переходів скінченного автомата при оформленні замовлення

На програмному рівні цей механізм забезпечується модулем управління станами, який кешує контекст кожного користувача в оперативній пам'яті сервера. Коли користувач ініціює процес покупки, система переводить його в активний стан очікування адреси. Усі наступні повідомлення перехоплюються спеціалізованим обробником, який ігнорує стандартні команди каталогу. Після отримання та внутрішньої перевірки тексту адреси, автомат здійснює транзакцію до стану очікування телефону. Лише після успішного проходження всього ланцюжка станів, тимчасовий словник контексту вивантажується з пам'яті, формується фінальний об'єкт замовлення і відбувається цілісний запис до бази даних. Життєвий цикл самого замовлення також підпорядковується логіці станів, які зберігаються у базі у вигляді цілочислових маркерів (в обробці, передано в доставку, виконано, скасовано). На рівні графічного інтерфейсу панелі адміністратора ці числові значення динамічно перетворюються у зрозумілі текстові мітки через функцію-маппер, що мінімізує обсяг збережених даних на диску.

2.4 Об'єктно-орієнтоване проєктування моделей взаємодії з даними

Для забезпечення структурної цілісності програмного комплексу, полегшення його подальшого масштабування та дотримання принципу чистої архітектури, всю логіку взаємодії з локальною базою даних інкапсульовано у класи відповідно до парадигм об'єктно-орієнтованого програмування [18]. Замість хаотичного розміщення SQL-запитів безпосередньо у функціях обробки повідомлень, що призвело б до високої зв'язності коду та ускладнило б рефакторинг, у системі розроблено набір моделей даних на основі патерну доступу до даних [19]. Кожна сутність предметної області представлена окремим класом, структура якого відображена на UML-діаграмі класів (рис. 2.4) [20].

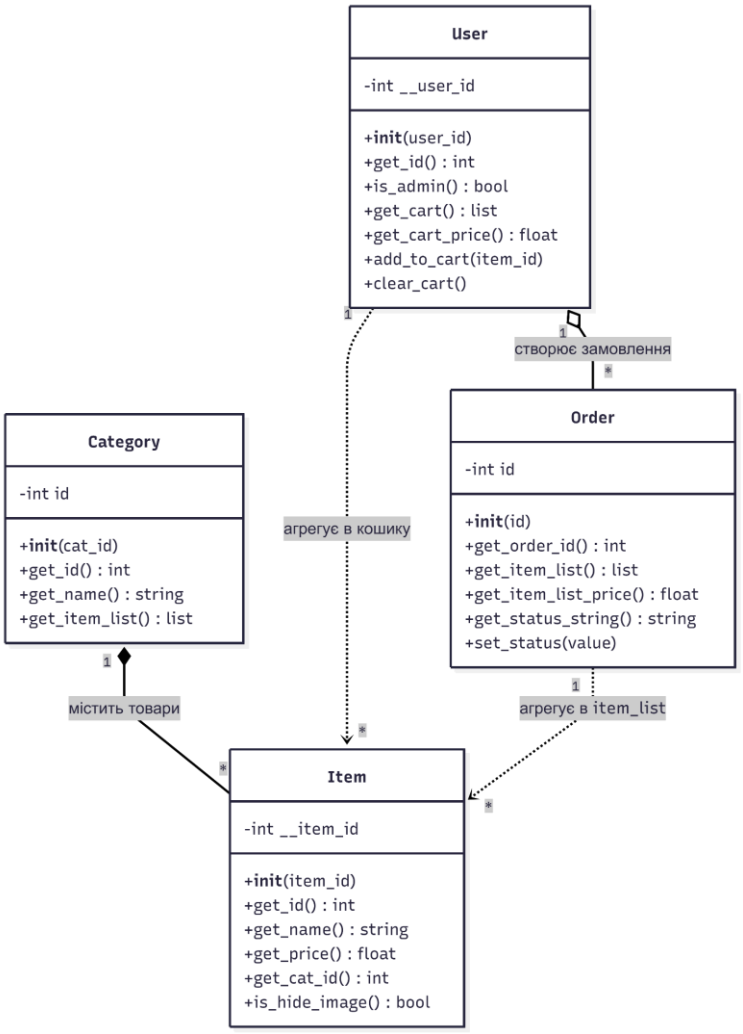


Рисунок 2.4 – UML-діаграма класів об'єктних моделей

Наприклад, при ініціалізації екземпляра класу користувача конструктор автоматично здійснює параметризований запит на перевірку наявності клієнта в системі. У разі відсутності відповідного запису, об'єкт самостійно ініціює операцію додавання нового клієнта, фіксуючи точний системний час першої взаємодії з ботом.

Завдяки об'єктному підходу, вся бізнес-логіка розрахунків також надійно прихована всередині методів відповідних класів додатку. Зокрема, метод розрахунку фінальної вартості кошика не вимагає від зовнішнього коду проведення обчислень. Він самостійно ітерує через десеріалізований список ідентифікаторів, звертається до екземплярів класу товарів для отримання їх актуальної ціни, підсумовує ці значення і автоматично інтегрує у фінальний

чек вартість логістичних послуг, якщо користувач попередньо обрав відповідну опцію доставки. Такий підхід забезпечує абсолютну ізоляцію рівня бізнес-логіки від рівня представлення (користувацького інтерфейсу месенджера). Завдяки цьому будь-які майбутні зміни в алгоритмах розрахунку ціноутворення або нарахування знижок вимагатимуть модифікації лише одного методу всередині відповідного класу, що гарантує високу стабільність системи та мінімізує ризики виникнення регресійних помилок.

2.5 Проєктування підсистем резервного копіювання та верифікації клієнтів

Оскільки розроблений інформаційний комплекс орієнтований на безперервну комерційну експлуатацію в реальних умовах, одним із найвищих пріоритетів при проєктуванні стала розробка автономних підсистем забезпечення цілісності даних та захисту від несанкціонованого доступу. Локальний характер розміщення бази даних забезпечує надзвичайно високу швидкість читання та запису, однак формує єдину точку відмови у вигляді фізичного файлу на накопичувачі. Для нівелювання цього ризику спроєктовано підсистему автоматизованого резервного копіювання. Даний програмний модуль за командою адміністратора або за встановленим системним розкладом генерує ізольовану директорію та здійснює безпечне копіювання бази даних разом із конфігураційними файлами. З метою запобігання переповненню дискового простору сервера імплементовано алгоритм ротації бекапів. Програма аналізує часові мітки створення архівів і безповоротно видаляє ті з них, вік яких перевищує встановлений ліміт. Крім того, розроблено механізм перевірки консистентності файлової системи, який періодично сканує директорію із зображеннями товарів та видаляє графічні файли, що більше не мають посилань у таблицях бази даних.

Не менш важливою складовою загальної архітектури безпеки є підсистема захисту від автоматизованих спам-ботів. Для запобігання штучному переповненню бази даних фіктивними реєстраціями, які можуть призвести до відмови в обслуговуванні, в логіку ініціалізації сесії вбудовано модуль графічної верифікації (CAPTCHA). Механізм працює за принципом динамічної генерації унікального растрового зображення із випадковою буквено-цифровою послідовністю. Дане зображення піддається алгоритмічним геометричним спотворенням для максимального ускладнення його аналізу системами оптичного розпізнавання тексту. Цей бар'єр діє як непереборний фільтр на нульовому етапі взаємодії. Лише після коректного введення користувачем верифікаційного коду, його унікальний ідентифікатор переноситься до бази даних довірених клієнтів, після чого система надає йому повний доступ до навігації каталогом товарів та оформлення замовлень.

3 ПРАКТИЧНА РЕАЛІЗАЦІЯ АСИНХРОННИХ МОДУЛІВ ТА ІНТЕРФЕЙСІВ СИСТЕМИ

3.1 Програмування ядра бота та обробки вхідних подій

Практична реалізація серверної частини програмного комплексу починається з ініціалізації ядра системи та налаштування механізмів маршрутизації вхідних подій. У головному виконуваному файлі додатку відбувається імпорт необхідних бібліотек та створення екземпляра класу Bot, який інкапсулює логіку взаємодії з Telegram API за допомогою унікального криптографічного токена. Цей токен не зашитий жорстко у вихідний код, а динамічно зчитується під час запуску через метод екземпляра класу Settings, що парсить зовнішній конфігураційний файл. Для управління чергою повідомлень та маршрутизації запитів ініціалізується об'єкт диспетчера (Dispatcher). Важливою архітектурною особливістю є підключення до диспетчера підсистеми MemoryStorage, яка виділяє ізольовані сегменти оперативної пам'яті сервера для збереження тимчасових контекстів діалогу кожного унікального користувача. Це забезпечує надзвичайно високу швидкість доступу до станів скінченного автомата без необхідності звернення до повільної дискової підсистеми.

Обробка вхідного потоку даних реалізована за допомогою системи декораторів, які прив'язують асинхронні функції-корутини до конкретних типів подій. Наприклад, базова функція привітання активується за допомогою декоратора обробки повідомлень, який реагує виключно на системну команду початку діалогу. При її виклику сервер створює екземпляр моделі User, передаючи конструктору ідентифікатор чату. Цей процес автоматично перевіряє наявність клієнта в базі даних SQLite та, за потреби, виконує реєстрацію. Після успішної авторизації клієнта ядро системи формує та відправляє вітальне повідомлення, текст якого динамічно підвантажується з

налаштувань, і прикріплює до нього базову клавіатуру з головним меню (ReplyKeyboardMarkup). Усі наступні текстові повідомлення, що не є командами, потрапляють до загального обробника текстового вводу, де за допомогою умовних конструкцій здійснюється виклик відповідних функціональних модулів, таких як виведення каталогу, відкриття кошика або звернення до розділу поширених запитань.

3.2 Розробка клієнтського функціоналу для навігації та управління кошиком

Користувацький інтерфейс системи побудований на концепції інлайн-клавіатур (InlineKeyboardMarkup), які прикріплюються безпосередньо до повідомлень і дозволяють клієнту взаємодіяти з каталогом без необхідності ручного введення тексту. При виборі розділу каталогу система формує запит до бази даних через об'єктну модель Category, отримує список усіх активних категорій та динамічно генерує кнопки, кожна з яких містить у собі прихований параметр зворотного виклику (callback_data). У головному циклі програми реалізовано універсальний обробник зворотних викликів, який перехоплює ці події. Залежно від префікса отриманих даних, обробник приймає рішення про подальші дії. Наприклад, якщо префікс вказує на вибір конкретної категорії, система звертається до методу вилучення списку товарів, пов'язаних із цією категорією, і замінює поточне повідомлення на оновлений список товарних позицій, що створює ефект безшовної навігації у межах одного екрана месенджера.

Особливу увагу в процесі розробки було приділено модулю управління віртуальним кошиком клієнта. Оскільки інформація про обрані товари зберігається у денормалізованому вигляді як текстовий рядок, додавання нової позиції реалізовано через конкатенацію. Коли клієнт натискає інлайн-кнопку додавання товару, обробник зворотного виклику викликає метод моделі User,

який отримує поточний рядок кошика, додає до нього новий ідентифікатор товару через кому та атомарно оновлює запис у базі даних. При переході до розділу кошика система зчитує цей рядок і виконує операцію десеріалізації. Програмний алгоритм розбиває рядок на масив, підраховує кількість входжень кожного унікального ідентифікатора та формує зведену таблицю обраних позицій. Вартість замовлення розраховується динамічно, помножуючи кількість кожної позиції на актуальну ціну, отриману з моделі товару. Цей підхід унеможливлює ситуації, коли ціна товару змінилася в базі, але у кошику клієнта залишилася старою, гарантуючи фінансову точність транзакцій. Візуальне представлення кошика з динамічно розрахованою сумою наведено на рисунку 3.1.



Рисунок 3.1 – Візуальне представлення кошика з динамічно розрахованою сумою

3.3 Реалізація механізмів збору контактної інформації при оформленні замовлень

Перехід клієнта від перегляду кошика до безпосереднього оформлення покупки активує підсистему управління станами (FSMContext). На цьому етапі

класична навігація через кнопки блокується, і система переводить сесію користувача в режим очікування введення специфічних даних. Першим кроком є запит на обрання способу доставки, що реалізовано через додаткову інлайн-клавіатуру. Якщо клієнт обирає кур'єрську доставку, стан сесії змінюється на очікування введення фізичної адреси. Текстове повідомлення клієнта перехоплюється виділеною корутиною, яка зберігає введену адресу у тимчасовому словнику оперативної пам'яті. Наступним кроком система запитує контактний номер телефону. Оскільки формат телефонних номерів є критично важливим для служби доставки, впроваджено жорстку валідацію вхідного рядка з використанням зовнішньої бібліотеки phonenumbers. Якщо користувач вводить некоректний набір цифр або пропускає код країни, бібліотека генерує виключення, яке перехоплюється системою. У результаті бот відправляє повідомлення про помилку і залишає користувача у поточному стані до моменту введення правильного формату.

Після успішного збору та валідації всіх необхідних атрибутів, система виконує транзакцію створення замовлення. Усі зібрані дані з тимчасового словника стану, а також десеріалізований масив товарів з кошика користувача, передаються до фабричного методу створення замовлення (create_order). Ця функція виконує атомарний SQL-запит на вставку нового запису в таблицю замовлень з первинним статусом обробки. Одразу після цього викликається метод очищення кошика користувача (clear_cart), який скидає текстове поле відповідного запису в базі даних до значення нуль-вказівника, звільняючи кошик для майбутніх покупок. Паралельно з цим, система використовує підсистему сповіщень для асинхронного розсилання повідомлень усім користувачам, які мають статус адміністратора, інформуючи їх про надходження нового замовлення з деталізацією вмісту та контактними даними покупця.

3.4 Програмне забезпечення функцій панелі адміністрування

Управління торговою площадкою вимагає наявності захищеного та зручного інструментарію. У розробленому комплексі панель адміністрування повністю інтегрована в інтерфейс Telegram-бота, доступ до якої регламентується суворою перевіркою прав (рольова модель доступу). При надходженні команди відкриття адмін-панелі ядро системи викликає метод перевірки привілеїв моделі User. Лише у випадку підтвердження статусу адміністратора система рендерить специфічну клавіатуру управління. Базовий функціонал адміністратора включає інструменти для роботи з категоріями та товарами. Наприклад, операція додавання нової категорії реалізована через ті самі механізми скінченного автомата. Система запитує назву нової категорії і після отримання відповіді викликає абстрактний метод створення, що здійснює запис у базу даних SQLite. Редагування існуючих товарів дозволяє змінювати їхню ціну, опис, прив'язку до категорії та прикріплювати нові зображення, які фізично зберігаються на локальному накопичувачі сервера. Приклад інтерфейсу управління товарним каталогом наведено на рисунку 3.2.



Рисунок 3.2 – Інтерфейс управління товарним каталогом

Окрім управління асортиментом, панель адміністратора містить критично важливі сервісні функції, що забезпечують стабільність системи. Одним із таких інструментів є модуль ручного ініціювання резервного копіювання. При виклику відповідної функції серверний скрипт звертається до бібліотек операційної системи для створення нової директорії, назва якої форматується за поточним системним часом. До цієї директорії виконується бінарне копіювання локального файлу бази даних та конфігураційних файлів INI. Також в адмін-панелі реалізовано інтерфейс для моніторингу статусу замовлень. Адміністратор отримує можливість переглядати список усіх активних замовлень, фільтрувати їх за числовими статусами та змінювати етапи їх виконання. Зміна статусу в базі даних автоматично генерує тригер, який відправляє клієнту сервісне повідомлення про те, що його замовлення, наприклад, передано кур'єру або успішно доставлено.

3.5 Створення підсистеми генерації візуальної аналітики

Прийняття управлінських рішень у сфері електронної комерції вимагає постійного моніторингу кількісних та фінансових показників. Оскільки базовий функціонал месенджерів не підтримує нативне виведення складних аналітичних дашбордів, у системі було спроєктовано та реалізовано автономний модуль програмної генерації візуальної статистики на базі математичної бібліотеки matplotlib. При запиті адміністратором статистичного звіту система ініціює складний алгоритм обробки даних. На першому етапі відбувається звернення до таблиці замовлень у базі даних, де відбираються всі успішно завершені транзакції за визначений часовий проміжок (наприклад, за останні 30 днів). Програмний модуль на мові Python ітерує через ці записи, десеріалізує масиви проданих товарів та агрегує загальну суму виручки і кількість реалізованих одиниць номенклатури у розрізі кожного дня.

На другому етапі агреговані масиви даних передаються до графічного ядра бібліотеки matplotlib. Ядро створює віртуальне полотно в оперативній пам'яті (буфер BytesIO), на якому здійснюється програмне малювання стовпчастих діаграм та графіків трендів. Стилістичне оформлення графіків, зокрема товщина ліній, розмір шрифту осей, кольорова палітра стовпчиків діаграм (визначена у шістнадцятковому форматі), динамічно підтягується з блоку налаштувань статистики зовнішнього конфігураційного файлу. Це дозволяє адміністраторам адаптувати візуальне представлення звітів без втручання у вихідний код програми. Після завершення рендерингу графічний буфер конвертується у стандартний формат растрового зображення та передається диспетчеру бота. Бот відправляє згенероване зображення адміністратору у вигляді фотографії з текстовим підписом ключових метрик, після чого віртуальний буфер очищується збирачем сміття (Garbage Collector), що запобігає витокам оперативної пам'яті сервера при масовій генерації звітів. Зразок згенерованого статистичного звіту представлено на рисунку 3.3.

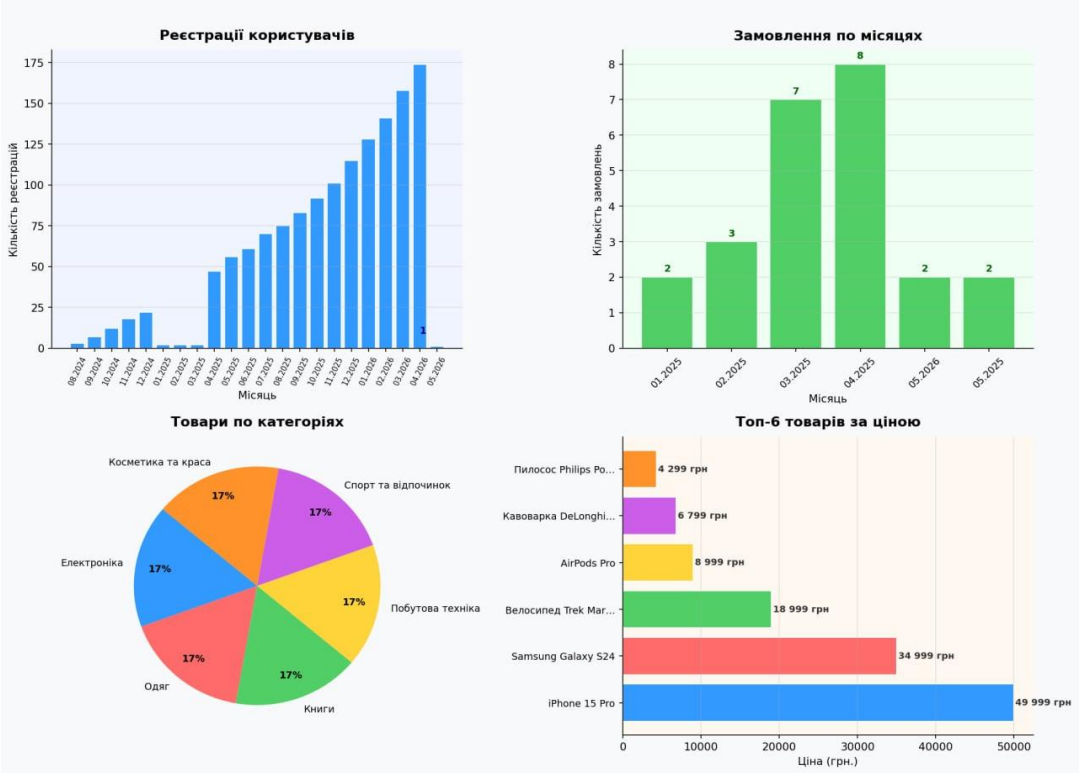


Рисунок 3.3 – Згенерований статистичний звіт

4 ЕКСПЛУАТАЦІЙНЕ ТЕСТУВАННЯ ТА ВПРОВАДЖЕННЯ ПРОГРАМНОГО ПРОДУКТУ

4.1 Верифікація роботи ключових алгоритмів та інтерфейсів

Процес тестування розробленого програмного комплексу є критично важливим етапом життєвого циклу розробки, який гарантує коректність виконання закладеної бізнес-логіки в умовах реальної експлуатації. Для перевірки працездатності ядра системи застосовувалися методи функціонального та інтеграційного тестування. Особлива увага приділялася верифікації роботи асинхронного диспетчера та правильності маршрутизації подій від інлайн-клавіатур. Під час тестування модуля управління кошиком моделювалися ситуації з одночасним або швидким послідовним додаванням великої кількості товарних позицій з різних категорій. Метою цього тестування було підтвердження коректності алгоритму денормалізації бази даних, зокрема процесу конкатенації ідентифікаторів товарів у єдиному текстовому полі SQLite. Перевірка підтвердила, що навіть при асинхронному надходженні запитів атомарність операцій оновлення бази даних зберігається, і втрати або пошкодження даних кошика не відбувається. Також ретельно тестувалася логіка скінченних автоматів при оформленні замовлення. Шляхом симуляції нестандартної поведінки користувача, наприклад, введення команд каталогу під час етапу очікування адреси доставки, було доведено, що система коректно утримує проміжний стан сесії в оперативній пам'яті і не дозволяє обійти обов'язкові кроки оформлення покупки.

4.2 Оцінка захищеності системи від некоректного введення даних

Надійність інформаційної системи визначається не лише її здатністю виконувати позитивні сценарії, але й здатністю протистояти некоректним або шкідливим вхідним даним. У ході експлуатаційного тестування проведено ґрунтовну оцінку підсистеми графічної верифікації користувачів. Моделювалися сценарії введення неправильних значень текстового коду з растрового зображення капчі. Результати підтвердили повне та безумовне блокування доступу до бізнес-логіки та каталогу для неавторизованих сесій, що нівелює загрозу автоматизованого парсингу товарів. Окремим критичним етапом стала перевірка модуля збору контактної інформації. За допомогою інтегрованої бібліотеки парсингу та перевірки форматів тестувалися випадки введення користувачами номерів телефонів у суто локальних форматах без використання обов'язкового міжнародного коду, а також спроби навмисного введення буквених чи спеціальних символів. У всіх змодельованих випадках система успішно перехоплювала виключення бібліотеки, блокувала транзакцію та повертала користувача на попередній крок введення з відповідним повідомленням про помилку. Для захисту від атак типу SQL-ін'єкцій на рівні підсистеми зберігання даних було перевірено імплементацію параметризованих запитів. Оскільки драйвер бази даних автоматично здійснює екранування всього вхідного тексту, введення користувачами шкідливих SQL-команд в поле адреси або коментаря розпізнається виключно як безпечний текстовий рядок. Додатково проведено стрес-тестування механізму перевірки цілісності файлової системи: при фізичному видаленні або перейменуванні файлу зображення товару безпосередньо на локальному диску, система коректно обробляла цю колізію та відправляла асинхронне системне сповіщення адміністраторам платформи без падіння загального процесу.

4.3 Підготовка середовища та розгортання бота на віддаленому сервері

Фінальним етапом реалізації проєкту стала підготовка серверного середовища та розгортання програмного продукту на віддаленому обчислювальному вузлі для забезпечення його цілодобової доступності. Для стабільної роботи комплексу було орендовано віртуальний приватний сервер під управлінням операційної системи сімейства Linux, що забезпечує гнучкий розподіл ресурсів та високий рівень мережевої безпеки. Процес розгортання розпочинався зі створення ізольованого віртуального середовища мови програмування за допомогою вбудованого модуля venv. Це дозволило відокремити програмні залежності бота від системних пакетів операційної системи, виключаючи можливі конфлікти версій. Далі за допомогою штатного менеджера пакетів було автоматизовано встановлено всі необхідні сторонні бібліотеки, перелік яких жорстко зафіксований у конфігураційному файлі залежностей проєкту.

Ключовим кроком налаштування продуктового середовища стало формування зовнішнього конфігураційного файлу системи, куди було внесено дійсний криптографічний токен Telegram API, згенерований платформою BotFather, та масив унікальних ідентифікаторів уповноважених адміністраторів. Для забезпечення безперервної роботи додатку у фоновому режимі та його автоматичного відновлення у випадку виникнення непередбачуваних критичних збоїв або планового перезавантаження сервера, було створено спеціалізований юніт-файл для системного менеджера Systemd. Це дозволило перевести основний скрипт виконання бота у статус фонового демона. Менеджер процесів здійснює безперервний моніторинг життєвого циклу додатку, записує стандартні потоки виводу та помилок до системного журналу, що суттєво спрощує подальше технічне обслуговування, аудит безпеки та моніторинг працездатності розробленої системи.

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи було успішно вирішено актуальну інженерну задачу – спроєктовано та програмно реалізовано високонавантажену інформаційну систему для електронної комерції у форматі Telegram-бота. Поставлена на початку дослідження мета була повністю досягнута через послідовне виконання комплексу теоретичних та практичних завдань. Створений програмний продукт демонструє високу ефективність у вирішенні проблем конверсії, притаманних класичним веб-орієнтованим інтернет-магазинам, завдяки перенесенню бізнес-процесів безпосередньо в інтерактивне середовище месенджера з мінімальним рівнем користувацького тертя.

Проведений на початковому етапі аналіз архітектурних підходів підтвердив необхідність відмови від застарілих синхронних моделей обробки мережевих запитів. Обґрунтовано вибір мови програмування Python у поєднанні з подієво-орієнтованим фреймворком aiogram, що дозволило реалізувати повноцінний асинхронний бекенд. Ця архітектура забезпечує здатність системи одночасно обробляти тисячі користувацьких сесій у межах єдиного системного потоку, ефективно вирішуючи проблему шторму повідомлень під час пікових навантажень. Стратегічне інженерне рішення відмовитися від ресурсомістких систем об’єктно-реляційного відображення на користь прямих оптимізованих параметризованих запитів до локальної бази даних SQLite гарантувало максимальну швидкодію підсистеми зберігання інформації.

У процесі проєктування логічної структури бази даних було успішно застосовано метод контрольованої денормалізації. Зберігання масивів обраних товарів у вигляді текстових рядків дозволило значно зменшити кількість ресурсомістких операцій об’єднання таблиць, перенісши тягар математичних обчислень на рівень прикладної логіки додатку. Для забезпечення

безперебійної маршрутизації складних багатокрокових діалогів під час оформлення замовлень було розроблено та імплементовано програмну модель скінченного автомата. Використання ізольованих сегментів оперативної пам’яті для кешування проміжних станів сесій повністю усунуло ризики втрати контексту спілкування та мінімізувало знос дискових накопичувачів сервера.

Практична цінність розробленого комплексу суттєво підвищується завдяки наявності закритої підсистеми адміністрування, яка дозволяє здійснювати повний цикл управління товарною номенклатурою та статусами замовлень безпосередньо через інтерфейс месенджера з дотриманням рольової моделі доступу. Інтеграція математичних інструментів для генерації візуальної статистики безпосередньо в оперативній пам’яті сервера забезпечує керівництво зручними аналітичними звітами без необхідності розгортання зовнішніх систем моніторингу. Впроваджені алгоритми захисту, зокрема система графічної верифікації клієнтів на базі динамічної капчі, жорстка валідація контактних даних за міжнародними стандартами та автономний модуль ротаційного резервного копіювання, гарантують високу надійність та стійкість системи до зовнішніх загроз.

Підсумовуючи результати експлуатаційного тестування та розгортання, можна стверджувати, що створена програмна система є повністю завершеним, безпечним та легко масштабованим інформаційним продуктом. Програмний комплекс не вимагає дорогих апаратних потужностей для своєї безперебійної роботи, легко конфігурується «на льоту» завдяки винесенню налаштувань у зовнішні файли і повністю готовий до комерційного впровадження у процеси роздрібної торгівлі як сучасна, економічно вигідна альтернатива класичним платформам електронної комерції.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Електронна комерція (e-commerce). Основні поняття та моделі. Вікіпедія : вільна енцикл. URL: https://uk.wikipedia.org/wiki/Електронна_комерція (дата звернення: 18.02.2026).

2. Офіційна документація Telegram Bot API. URL: https://core.telegram.org/bots/api (дата звернення: 12.03.2026).

3. Розмовна комерція (Conversational Commerce) у сучасних месенджерах. URL: https://en.wikipedia.org/wiki/Conversational_commerce (дата звернення: 20.03.2026).

4. Документація мови програмування Python. URL: https://docs.python.org/3/ (дата звернення: 10.03.2026).

5. PEP 8. Керівництво щодо стилю написання коду на Python. URL: https://peps.python.org/pep-0008/ (дата звернення: 22.03.2026).

6. Документація асинхронного фреймворку aiogram. URL: https://docs.aiogram.dev/en/latest/ (дата звернення: 15.03.2026).

7. Бібліотека asyncio. Асинхронний ввід-вивід. Документація Python. URL: https://docs.python.org/3/library/asyncio.html (дата звернення: 20.03.2026).

8. Офіційна документація СУБД SQLite. URL: https://www.sqlite.org/docs.html (дата звернення: 18.03.2026).

9. Модуль sqlite3. Інтерфейс бази даних SQLite для Python. URL: https://docs.python.org/3/library/sqlite3.html (дата звернення: 22.03.2026).

10. Реляційні бази даних та мова SQL. Навчальні матеріали W3Schools. URL: https://www.w3schools.com/sql/ (дата звернення: 12.04.2026).

11. Модуль configparser. Документація Python. URL: https://docs.python.org/3/library/configparser.html (дата звернення: 25.03.2026).

12. Бібліотека phonenumbers. Обробка, форматування та валідація телефонних номерів. URL: https://pypi.org/project/phonenumbers/ (дата звернення: 02.04.2026).

13. Бібліотека captcha. Генерація графічних контрольних зображень. URL: https://pypi.org/project/captcha/ (дата звернення: 05.04.2026).

14. Бібліотека Matplotlib. Візуалізація даних у Python. URL: https://matplotlib.org/stable/users/index.html (дата звернення: 28.03.2026).

15. Технологія Long Polling для клієнт-серверної взаємодії. Вікіпедія : вільна енцикл. URL: https://en.wikipedia.org/wiki/Push_technology#Long_polling (дата звернення: 16.04.2026).

16. Нормалізація та денормалізація баз даних. Вікіпедія : вільна енцикл. URL: https://uk.wikipedia.org/wiki/Нормалізація_баз_даних (дата звернення: 14.04.2026).

17. Теорія скінченних автоматів (Finite State Machine). Вікіпедія : вільна енцикл. URL: https://uk.wikipedia.org/wiki/Скінченний_автомат (дата звернення: 10.04.2026).

18. Об'єктно-орієнтоване програмування (ООП). Вікіпедія : вільна енцикл. URL: https://uk.wikipedia.org/wiki/Об'єктно-орієнтоване_програмування (дата звернення: 25.03.2026).

19. Патерн проєктування Data Access Object (DAO). Вікіпедія : вільна енцикл. URL: https://en.wikipedia.org/wiki/Data_access_object (дата звернення: 08.04.2026).

20. Уніфікована мова моделювання (UML). Побудова діаграм прецедентів та класів. URL: https://uk.wikipedia.org/wiki/UML (дата звернення: 28.03.2026).

Декларація

академічної доброчесності

здобувача освіти ВСП «Економіко-правничий фаховий коледж ЗНУ»

Я, Скочко Данило Олександрович, здобувач освіти 4 курсу, спеціальності 121 Інженерія програмного забезпечення, освітньо-професійної програми Розробка програмного забезпечення, групи К121-22, адреса електронної пошти danillolerr@gmail.com

- підтверджую, що написана мною дипломна робота на тему «Проєктування та розробка системи продажу цифрових товарів у Telegram-боті» відповідає вимогам академічної доброчесності та не містить порушень, що визначені у ст. 42 Закону України «Про освіту», зі змістом яких ознайомлений/ознайомлена;

- заявляю, що надана мною для перевірки електронна версія роботи є ідентичною її друкованій версії;

- згоден/згодна на перевірку моєї роботи на відповідність критеріям академічної доброчесності у будь-який спосіб, у тому числі за допомогою інтернет-системи, а також на архівування моєї роботи в базі даних цієї системи.

Дата 27.05.2026 Підпис  Данило СКОЧКО

Ім'я ПРІЗВИЩЕ (здобувача освіти)

Дата _____ Підпис _____ Дмитро ДРАЄВСЬКИЙ

Ім'я ПРІЗВИЩЕ (наукового керівника)